Devonfw Methodology

2017-03-30

Table of Contents

1. System Specification

There are many methods to write specifications, especially business-oriented system specifications. As a part of the devonfw methodology module, one such method is described, which is pragmatic, easy to use and open for every team member.

This method is described in the following ways:

Guide
: As a guide to the creation of a system specification and the integration into an agile project methodology

Example
: As an example of a system specification, written for the restaurant example of devonfw.

include::System-Specification-Guide

include::System-Specification-Example

2. Client Architecture

2.1 OASP / Devon Client Architecture

2.1.1 Introduction

Purpose of this document

In our business applications, the clients often are much more complex to develop and design than the server. Nonetheless, where we have a concrete layered technical architecture for the server in OASP, we are still lacking a pendant on the client, where we only define to have something called dialog components. Finding an concrete architecture applicable for all clients may on the other hand be difficult to accomplish.

This document tries to define on a high abstract level, a reference architecture which is supposed to be a mental image and frame for orientation regarding the evaluation and appliance of different client frameworks. As such it defines terms and concepts required to be provided for in any framework and thus gives a common ground of understanding for those acquainted with the reference architecture. This allows better comparison between the various frameworks out there, each having their own terms for essentially the same concepts. It also means that for each framework we need to explicitly map how it implements the concepts defined in this document.

The architecture proposed herein is neither new nor was it developed from scratch. Instead it is the gathered and consolidated knowledge and best practices of various projects (s. References).

Goal of the Client Architecture

The goal of the client architecture is to support the non-functional requirements for the client, i.e. mostly maintainability, scalability, efficiency and portability. As such it provides a component-oriented architecture following the same principles listed already in the OASP architecture overview. Furthermore it ensures a homogeneity regarding how different concrete UI technologies are being applied in the projects, solving the common requirements in the same way.

Architecture Views

As for the server we distinguish between the business and the technical architecture. Where the business architecture is different from project to project and relates to the concrete design of dialog components given concrete requirements, the technical architecture can be applied to multiple projects.

The focus of this document is to provide a technical reference architecture on the client on a very abstract level defining required layers and components. How the architecture is implemented has to be defined for each UI technology.

The technical infrastructure architecture is out of scope for this document and although it needs to be considered, the concepts of the reference architecture should work across multiple TI architecture, i.e. native or web clients.

2.1.2 OASP Reference Client Architecture

The following gives a complete overview of the proposed reference architecture. It will be built up incrementally in the following sections.

Figure 1 Overview

Client Architecture

On the highest level of abstraction we see the need to differentiate between dialog components and their container they are managed in, as well as the access to the application server being the backend for the client (e.g. an OASP4J instance). This section gives a summary of these components and how they relate to each other. Detailed architectures for each component will be supplied in subsequent sections

Figure 2 Overview of Client Architecture

Dialog Component

A dialog component is a logical, self-contained part of the user interface. It accepts user input and actions and controls communication with the user. Dialog components use the services provided by the dialog container in order to execute the business logic. They are self-contained, i.e. they possess their own user interface together with the associated logic, data and states.

- Dialog components can be composed of other dialog components forming a hierarchy
- Dialog components can interact with each other. This includes communication of a parent to its children, but also between components independent of each other regarding the hierarchy.

Dialog Container

Dialog components need to be managed in their lifecycle and how they can be coupled to each other. The dialog container is responsible for this along with the following:

- Bootstrapping the client application and environment
  - Configuration of the client
  - Initialization of the application server access component
- Dialog Component Management
  - Controlling the lifecycle
  - Controlling the dialog flow
  - Providing means of interaction between the dialogs
  - Providing application server access
  - Providing services to the dialog components
    (e.g. printing, caching, data storage)
- Shutdown of the application

Application Server Access

Dialogs will require a backend application server in order to execute their business logic. Typically in an OASP application the service layer will provide interfaces for the functionality exposed to the client. These business oriented interfaces should also be present on the client backed by a proxy handling the concrete call of the server over the network. This component provides the set of interfaces as well as the proxy.

Dialog Container Architecture

The dialog container can be further structured into the following components with their respective tasks described in own sections:

Figure 3 Dialog Container Architecture

Application

The application component represents the overall client in our architecture. It is responsible for bootstrapping all other components and connecting them with each other. As such it initializes the components below and provides an environment for them to work in.

Configuration Management

The configuration management manages the configuration of the client, so the client can be deployed in different environments. This includes configuration of the concrete application server to be called or any other environment-specific property.

Dialog Management

The Dialog Management component provides the means to define, create and destroy dialog components. It therefore offers basic lifecycle capabilities for a component. In addition it also allows composition of dialog components in a hierarchy. The lifecycle is then managed along the hierarchy, meaning when creating/destroying a parent dialog, this affects all child components, which are created/ destroyed as well.

Service Registry

Apart from dialog components, a client application also consists of services offered to these. A service can thereby encompass among others:

- Access to the application server
- Access to the dialog container functions for managing dialogs or accessing the configuration
- Dialog independent client functionality such as Printing, Caching, Logging, Encapsulated business logic such as tax calculation
- Dialog component interaction

The service registry offers the possibility to define, register and lookup these services. Note that these services could be dependent on the dialog hierarchy, meaning different child instances could obtain different instances / implementations of a service via the service registry, depending on which service implementations are registered by the parents.

Services should be defined as interfaces allowing for different implementations and thus loose coupling.

Dialog Component Architecture

A dialog component has to support all or a subset of the following tasks:
(T1) Displaying the user interface incl. internationalization
(T2) Displaying business data incl. changes made to the data due to user interactions and localization of the data
(T3) Accepting user input including possible conversion from e.g. entered Text to an Integer
(T4) Displaying the dialog state

(T5) Validation of user input
(T6) Managing the business data incl. business logic altering it due to user interactions
(T7) Execution of user interactions
(T8) Managing the state of the dialog (e.g. Edit vs. View)
(T9) Calling the application server in the course of user interactions

Following the principle of separation of concerns, we further structure a dialog component in an own architecture allowing us the distribute responsibility for these tasks along the defined components:

Figure 4 Overview of dialog component architecture

Presentation Layer

The presentation layer generates and displays the user interface, accepts user input and user actions and binds these to the dialog core layer (T1-5). The tasks of the presentation layer fall into two categories:

- **Provision of the visual representation (View component)**
  The presentation layer generates and displays the user interface and accepts user input and user actions. The logical processing of the data, actions and states is performed in the dialog core layer. The data and user interface are displayed in localized and internationalized form.

- **Binding of the visual representation to the dialog core layer**
  The presentation layer itself does not contain any dialog logic. The data or actions entered by the user are then processed in the dialog core layer. There are three aspects to the binding to the dialog core layer. We refer to ???data binding???, ???state binding??? and ???action binding???. Syntactical and (to a certain extent) semantic validations are performed during data binding (e.g. cross-field plausibility checks). Furthermore, the formatted, localized data in the presentation layer is converted into the presentation-independent, neutral data in the dialog core layer (parsing) and vice versa (formatting).

Dialog Core Layer

The dialog core layer contains the business logic, the control logic, and the logical state of the dialog. It therefore covers tasks T5-9:

- **Maintenance of the logical dialog state and the logical data**
  The dialog core layer maintains the logical dialog state and the logical data in a form which is independent of the presentation. The states of the presentation (e.g. individual widgets) must not be maintained in the dialog core layer, e.g. the view state could lead to multiple presentation states disabling all editable widgets on the view.

- **Implementation of the dialog and dialog control logic**
  The component parts in the dialog core layer implement the client specific business logic and the dialog control logic. This includes, for example, the manipulation of dialog data and dialog states as well as the opening and closing of dialogs.

- **Communication with the application server**
  The dialog core layer calls the interfaces of the application server via the application server access component services.

The dialog core layer should not depend on the presentation layer enforcing a strict layering and thus minimizing dependencies.

Interactions between dialog components

Dialog components can interact in the following ways:

- **Embedding of dialog components**
  As already said dialog components can be hierarchically composed. This composition works by embedding on dialog component within the other. Apart from the lifecycle managed by the dialog container, the embedding needs to cope for the visual embedding of the presentation and core layer.

  - **Embedding dialog presentation**
    The parent dialog needs to either integrate the embedded dialog in its layout or open it in an own model window.

  - **Embedding dialog core**
    The parent dialog needs to be able to access the embedded instance of its children. This allows initializing and changing their data and states. On the other hand the children might require context information offered by the parent dialog by registering services in the hierarchical service registry.

- **Dialog flow**
  Apart from the embedding of dialog components representing a tight coupling, dialogs can interact with each other by passing the control of the UI, i.e. switching from one dialog to another.

When interacting, dialog components should interact only between the same or lower layers, i.e. the dialog core should not access the presentation layer of another dialog component.

2.1.3 Appendix

Notes about Quasar Client

The Quasar client architecture as the consolidated knowledge of our CSD projects is the major source for the above drafted architecture. However, the above is a much simplified and more agile version thereof:

- Quasar Client tried to abstract from the concrete UI library being used, so it could decouple the business from the technical logic of a dialog. The presentation layer should be the only one knowing the concrete UI framework used. This level of abstraction was dropped in this reference architecture, although it might of course still make sense in some projects. For fast-moving agile projects in the web however introducing such a level of abstraction takes effort with little gained benefits. With frameworks like Angular 2 we would even introduce one additional seemingly artificial and redundant layer, since it already separates the dialog core from its presentation.

- In the past and in the days of Struts, JSF, etc. the concept of session handling was important for the client since part of the client was sitting on a server with a session relating it to its remote counterpart on the users PC. Quasar Client catered for this need, by very prominently differentiating between session and application in the root of the dialog component hierarchy. However, in the current days of SPA applications and the lowered importance of servers-side web clients, this prominent differentiation was dropped. When still needed the referenced documents will provide in more detail how to tailor the respective architecture to this end.

2.1.4 References

- Architecture Guidelines for Application Design: https://troom.capgemini.com/sites/vcc/engineering/Cross%20Cutting/ArchitectureGuide/Architecture_Guidelines_for_Application_Design_v2.0.docx

- Quasar Client Architekturen: https://troom.capgemini.com/sites/vcc/Shared%20Documents/CrossCuttingContent/TopicOrientedCCC/QuasarOverview/NCE%20Quasar%20Review%20Workshop%202009-11-17/Quasar%20Development/Quasar-Client-Architectures.doc

Unresolved directive in DevonfwMethodology.asciidoc - include::OASP-Angular-Architecture.asciidoc[]